Hybrid recommender system

Dimopoulou Ivi, Dona Ergys, Prasopoulos Konstantinos
Department of Computer Science, EPFL, Switzerland

***Abstract*—In this report we describe the process that we followed, in order to be able to predict the rating that a specific person (or user) would give to a specific movie (or item), based on a dataset of known ratings. We present the various unsupervised ML models with which we experimented, as well as a supervised blending technique in which a variety of models are combined with results better than the best of the models. We evaluate each technique's performance and explain the observations that we made regarding their behaviour. Our best approach achieves a Root Mean Square Error (RMSE) of 1.017 on CrowdAI and 1.0155 locally.**

I. Data Preprocessing

The first step of our approach was the exploration of the dataset to identify potential issues and opportunities for data engineering. In order to do that, we created two charts: one for the number of ratings per user and one for the number of ratings per item, in order to determine whether there are any outliers or discrepancies in the data. As we can see in figure 1, the data seem relatively clean and the user and item ratings follow a somewhat expected tendency - few users have rated a lot of items, while more users have rated only a few items, and few items have been rated multiple times, while more items have been rated only a few times.

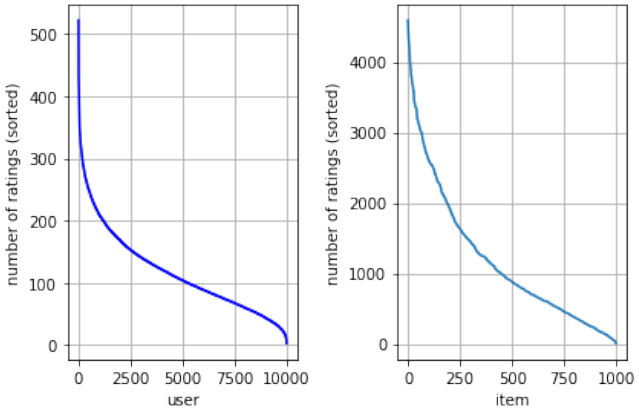

Figure 1: Distribution of ratings per user and per item

The types of algorithms that we have used and are describing below, do not predispose us to split the data in any way, given that they rely on as much known data as they can get, in order to predict unknown ratings. Splitting the data into parts and training each part with a separate model, would cause each model to have less known data to rely on and thus would most probably result in less accurate predictions. For this reason, we decided to not split or preprocess our data.

II. Tuning, training and evaluation process

A. Tuning

In order to tune our models and obtain the most reliable results possible, we used the standard technique of grid search among all the possible values of the hyper-parameters of each model, together with k-fold cross-validation with $k = 3$, for the evaluation of each value combination. The metric that was used for the evaluation is the Root Mean Square Error (RMSE) between the models' predictions on a specified test set and the true ratings of the test set. In the following paragraphs, we will use the following notation to denote specific hyper-parameters (where applicable):

- K: number of latent features
- N: maximum number of neighbours
- γ: learning rate
- λ: regularizer
- n_epochs: number of iterations
- n_cltr_u: number of user clusters
- n_cltr_i: number of item clusters

B. Training and evaluation

In order to blend the unsupervised models, we split our known data to a training set and a test set. The training set is used to train them and the test set is used to blend them. The splitting approach that we used gives a fairly accurate estimation of each model's real error. We iterate over all users in our known ratings, and randomly select 10% of each user's ratings to put in our test set, while leaving the other 90% in the training set. This way, as far as the number of ratings per user is concerned, both our training and test set have the same statistical properties as the original set of known ratings. More specifically, the number of ratings per user, both in the training and the test set, follows the same distribution as in the original dataset. This helps our local estimation of the test error be fairly accurate, compared to the real error that we get when we submit our predictions on CrowdAI.

III. Model Evaluation

A. Baseline

The model that we used as a baseline for our comparisons, is the Item Mean which is a simple model that calculates

the mean value of each item's known ratings and returns this value as the prediction of all item's unknown ratings, regardless of user. The model achieves a local test error of 1.0676 (RMSE).

B. Other models

During our experimentation, we made use of a plethora of models, which we have compared with our baseline model, as well as among each other. The models and their performance are briefly described below:

1) User mean: This model calculates the mean of each user's known ratings, and returns this value as the prediction of the user's unknown ratings, regardless of item. This model yielded a local test error of 1.1166.

2) Matrix factorisation with ALS: This model implements the well-known Matrix Factorisation algorithm, using Least Squares [1]. We have made use of Spark's [2] implementation of the algorithm. By tuning the algorithm, we concluded that the best hyper-parameters for the algorithm were: $\lambda = 0.09$ and $K = 10$. The algorithm gave a local test error of 1.0237.

Figure 2 nicely illustrates the role of each hyper-parameter. In the left case we can see that as K increases the disparity between the testing and training error also increases suggesting overfitting. In the right figure, the increase of the lambda parameters tends to reduce overfitting but also penalises the test score after some point.

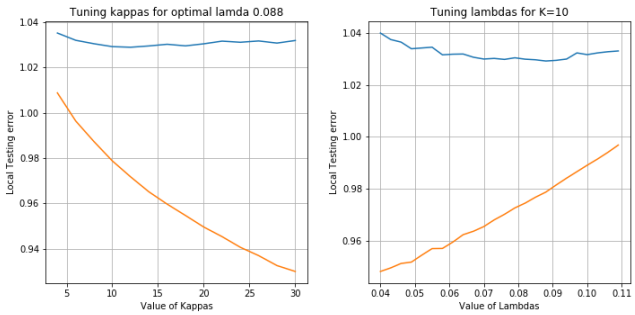

Figure 2: Blue: Testing error, Orange: Training error

3) SVD with Gradient Descent: This model implements a recommender algorithm using svd [3]. The implementation belongs to the Surprise library for recommender systems [4]. Using the following hyper-parameters: $K = 10$, $n\_epochs = 400$, $\gamma = 0.0002$ and $\lambda = 0.009$, the algorithm gives a local test error of 1.04311 which is only slightly better than the baseline.

4) SVD++ with Gradient Descent: The SVD++ model is an extension of the SVD algorithm [3], implemented by Surprise. We used the same hyper-parameters as with plain SVD, and achieved a slightly better local test error of 1.0324.

5) KNN inspired algorithms: KNN inspired algorithms are very widely used in the area of recommender systems. We have experimented with both user-based KNN algorithms, which calculate the similarity between users and provide predictions of a user's ratings, based on what similar users have rated, as well as item-based algorithms, which calculate the similarity between items, and provide predictions of a user's rating for an item, based on the ratings that the user has given on similar items. More specifically, we have used both user-based and item-based KNN with Means, KNN with Baseline and KNN with ZScore, provided by the Surprise library.

The number N of neighbours that we used for each algorithm is 900 for user-based KNN with Means, 400 for user-based KNN with Baseline and 90 for item-based KNN with Means, item-based KNN with Baseline and item-based KNN with ZScore. Generally, item-based algorithms have performed better than user-based algorithms, as can be seen in table I, which could be due to the fact that items are fewer than users in our dataset of known ratings. Therefore, the dataset contains more known ratings per item, on which item-based algorithms can rely, rather than known ratings per user, which user-based algorithms rely on.

6) Slope One: Another model with which we experimented, is Surprise's Slope One [5], which achieved a local test error of 1.0379.

7) Co-Clustering: Finally, we made use of a clustering technique called Co-Clustering [6], provided by Surprise, with hyper-parameters: $n\_epochs = 150$, $n\_cltr\_u = 2$, $n\_cltr\_i = 7$, which achieved a local test error of 1.0408.

Model	**Test RMSE**	**Hyper-parameters**
User Mean	1.1166	-
Item Mean	1.0676	-
ALS	1.0237	$\lambda = 0.09$, $K = 10$
SVD	1.0431	$K = 10$, $n\_epochs = 400$, $\gamma = 0.0002$, $\lambda = 0.009$
SVD++	1.0324	$K = 10$, $n\_epochs = 400$, $\gamma = 0.0002$, $\lambda = 0.009$
User-based with Means	1.0289	$N = 900$
User-based with Baseline	1.0300	$N = 400$
Item-Based with Means	1.0289	$N = 90$
Item-based with Baseline	1.0256	$N = 90$
Item-based with ZScore	1.0288	$N = 90$
Slope One	1.0379	-
Co-Clustering	1.0408	$n_epochs = 150$, $n\_cltr\_u = 2$, $n\_cltr\_i = 7$
Blend	1.0155	$\lambda = 0.001$

Table I: Local testing RMSE of all unsupervised models and of their blending

In general, the models which had the most accurate predictions, were matrix factorisation with ALS and item-based KNN with Baseline.

C. Model blending

Although some of the models with which we have experimented perform fairly well (e.g. Matrix Factorisation with ALS and item-based KNN with Baseline), our final approach consists of a blend of the previously described models that achieves a significantly lower RMSE. This approach was inspired by the one used by the winners of the 2009 Netflix Prize [7]. The main idea behind the technique is that, using the predictions of a set of different ML models for the same user-item pair, and creating a linear combination of all the predictions, each of which is multiplied by a weight factor, calculated based on how well each model predicts a subset of the known ratings, will lead to better prediction accuracy than the one achieved by each model separately. The technique indeed performs very well in practice, as is visible in table I.

The process that we followed in order to achieve the most optimal blending, first uses the various **unsupervised** learning techniques presented above, which are evaluated based on a known part of the dataset that is not used for training ("test-set"), as described in section II.B. Then, it uses **supervised** learning to blend the predictions that the trained models make, based on how well they predict this test set.
The process is described in the following steps:

1) Tune all models D individually, as described in section II-A.
2) Split the known data into 90% training set - 10% test set (y) which is a vector of ratings (n ratings).
3) Train all the models with the training set.
4) For all possible combinations c between available models:
   a) Create matrix X ((n, D)), where each column corresponds to the predictions that a model made for the real ratings y ($(n, 1)$). The predictions in each column are aligned to match the equivalent values of y
   b) Use Ridge Regression to calculate a weight vector w ($(D, 1)$) that minimises RMSE between $X \cdot w$ and y
   c) If RMSE is the best seen so far, store c as $c*$ and w as $w*$
5) For all models in combination $c*$:
   a) Train model using entire dataset of known ratings
6) Produce matrix X' ((n, D)), where
   rows: ratings of sample submission file
   columns: models of the combination
   cells: prediction of corresponding model for corresponding rating of sample submission file
7) Return final submission: $X' \cdot w*$ (essentially multiplying each model with the corresponding weight).

It is important to note, that the final blend, does not necessarily contain all of the available algorithms, but instead, the combination of models that achieves the best local RMSE on the test set. This happens, in part, because we use ridge regression, that affects the weights in ways not only tied to the accurate prediction of y (high w values are penalised to reduce overfitting). We also experimented with a variety of other minimisation functions including Least Squares. The latter in particular yielded a local error of just 1.0146 which was clearly overfitting the test set that we used to produce the weights, since the CrowdAI score got worse, compared to the one obtained with Ridge Regression. We therefore decided to use Ridge Regression, as it is the most robust choice that better matches the CrowdAI scores and also guarantees that the $X - \lambda I$ matrix is invertible.

Model	**Weight**
User Mean	-0.0310
Item Mean	0.0279
ALS	0.7165
SVD	-0.0118
SVD++	0.1677
User-based with Means	-0.0311
User-based with Baseline	0.2237
Item-Based with Means	-0.1048
Item-based with Baseline	0.3849
Slope One	-0.3130
Co-Clustering	-0.0450

Table II: Weights given to each model by ridge regression

Interestingly, the unsupervised-and-then-supervised training approach seems to work well even while using a single model. Concretely, when training ALS against the whole dataset (unsupervised only) we get an RMSE of 1.025 on CrowdAI. But, if we instead first perform unsupervised training on 90% of the dataset, then use the last 10% to train a single weight (w shape is $(1, 1)$), using Ridge Regression as described in the algorithm, and then apply the weight to the predictions of ALS that was trained on the whole dataset, the RMSE drops to 1.019 locally and to 1.02 on CrowdAI.

In figure 3 we can see how the number of models included in the blend affects the obtained rmse (the best combination per number of models is shown).

The best blend that we have achieved, has a local RMSE of 1.0155 and an RMSE of 1.017 on CrowdAI, surpassing the individual error of each of the models presented above. The participating models and their corresponding weights can be seen in Table II. The weights are not restricted to add up to 0 but they tend to fall close to it. It is interesting that even individually bad models such as Slope One help reduce the error of the blend by getting a negative weight.

IV. Conclusion

The conclusion that can be drawn, is the following: do not underestimate under-performing models, as they can inform better models about what they should not do. The

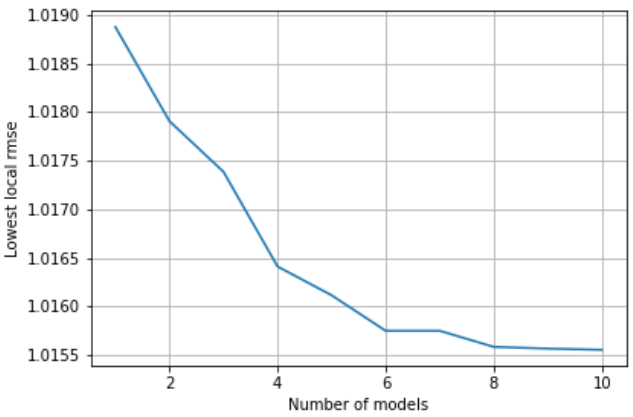

Figure 3: The lowest RMSE achievable by combining n models at a time

combination of a variety of unsupervised models yields results that are significantly better than any of the models individually. A potential future improvement of this work would be to incorporate cross-validation in the blending algorithm, so that overfitting on a specific part of the data is avoided.

REFERENCES

[1] Y. Koren, R. Bell, and C. Volinsky, "Matrix factorization techniques for recommender systems," *Computer*, vol. 42, no. 8, pp. 30–37, Aug. 2009. [Online]. Available: http://dx.doi.org/10.1109/MC.2009.263

[2] X. Meng, J. Bradley, B. Yavuz, E. Sparks, S. Venkataraman, D. Liu, J. Freeman, D. Tsai, M. Amde, S. Owen, D. Xin, R. Xin, M. J. Franklin, R. Zadeh, M. Zaharia, and A. Talwalkar, "Mllib: Machine learning in apache spark," *J. Mach. Learn. Res.*, vol. 17, no. 1, pp. 1235–1241, Jan. 2016. [Online]. Available: http://dl.acm.org/citation.cfm?id=2946645.2946679

[3] *Recommender Systems Handbook*. Springer, 2010. [Online]. Available: https://www.amazon.com/Recommender-Systems-Handbook-Francesco-Ricci-ebook/dp/B00D8D1Y10?SubscriptionId=AKIAIOBINVZYXZQZ2U3A&tag=chimbori05-20&linkCode=xm2&camp=2025&creative=165953&creativeASIN=B00D8D1Y10

[4] N. Hug, "Surprise, a Python library for recommender systems," http://surpriselib.com, 2017.

[5] D. Lemire and A. Maclachlan, "Slope one predictors for online rating-based collaborative filtering," *Proceedings of the 2005 SIAM International Conference on Data Mining*, Apr 2005. [Online]. Available: http://dx.doi.org/10.1137/1.9781611972757.43

[6] T. George and S. Merugu, "A scalable collaborative filtering framework based on co-clustering," in *Proceedings of the Fifth IEEE International Conference on Data Mining*, ser. ICDM '05. Washington, DC, USA: IEEE Computer Society, 2005, pp. 625–628. [Online]. Available: http://dx.doi.org/10.1109/ICDM.2005.14

[7] A. Tscher and M. Jahrer, "The bigchaos solution to the netflix grand prize," 01 2009.